

Parcours Ingénieur Intelligence Artificielle



- Projet 9 -

**Réalisez une application de
recommandation de contenu**



My Content

Romain Le Goff
31/08/2022


Liens

GitHub : https://github.com/Deviluna29/oc_ingenieur-ia_P9

Application Web : <https://ocp9-recommandation.azurewebsites.net/>



Sommaire

- 
1. Contexte
 2. Présentation du jeu de données
 3. Modélisation
 4. Architecture serverless
 5. Déploiement
 6. Architecture cible
 7. Conclusion



Contexte



- **My Content** est une start-up qui veut encourager la lecture en recommandant des contenus pertinents pour ses utilisateurs.
- La start-up est en pleine construction d'un **premier MVP** qui prendra la forme d'une **application**, qui consiste en une solution de **recommandation d'articles** et de livres à des particuliers.

- **Les objectifs :**

- Utiliser un jeu de données disponible en ligne pour développer le MVP. (interactions des utilisateurs avec les articles)
- Entraînement, tests et comparaison de plusieurs **modèles de recommandation**
- Mettre en place une **architecture serverless** pour un déploiement en production.



Présentation du jeu de données

- Le jeu de données provient de Globo.com, il contient les fichiers suivants :
 - `articles_metadata.csv` : fichier contenant des informations sur tous les articles publiés
 - `articles_embeddings.pickle` : fichier contenant un embedding de tous les articles
 - `clicks_sample.csv` : fichier contenant un échantillon des interactions des sessions des utilisateurs
 - `/clicks` : dossier contenant toutes les interactions des sessions des utilisateurs (un fichier par heure)
- 364 047 articles
- 2 988 181 clics
- 322 897 utilisateurs
- 250 features d'embedding du contenu des articles

Modélisation

On va ici présenter les modèles qui seront utilisés. Il y a deux approches possibles pour recommander des articles aux utilisateurs :

- **Content-Based :**

Ce modèle se base sur les préférences de l'utilisateur, en recommandant des articles similaires aux articles qu'il a déjà lu.

- + Le modèle n'a pas besoin de données sur les autres utilisateurs
Peut capturer les intérêts spécifiques d'un utilisateur
- Difficulté à représenter les caractéristiques des éléments (ici embedding déjà fourni)
Dépendant du nbr d'articles lus par l'utilisateur, et ne peut deviner d'autres préférences possibles

- **Collaborative-Filtering :**

Ce modèle se base sur les préférences des autres utilisateurs ayant lus les mêmes articles, en recommandant des articles lus par les autres utilisateurs aux préférences communes.

- + Peut aider l'utilisateur à découvrir de nouveaux intérêts
Pas besoin de connaissance sur le domaine (ici les articles), embedding automatiquement appris
- Difficulté à recommander les nouveaux éléments, car peu ou pas d'interaction avec ceux ci

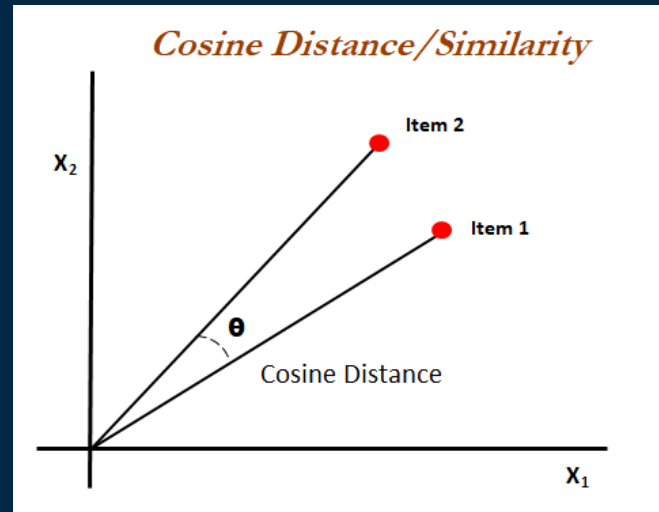
Modélisation

Content-Based :

- Le modèle utilise le calcul du **cosinus** entre les articles lus par l'utilisateur et les autres articles, qui sont projetés via leurs features de la matrice d'embedding dans un espace vectoriel (de dimension 250).
- La prédiction du modèle retourne les n plus proches articles par rapport aux articles déjà lus.

ACP :

- Le fichier d'embedding étant volumineux, il faudrait l'alléger pour une utilisation future dans le Cloud.
- L'ACP permet de passer de 250 features à 70, et de garder une variance de 0.977.
- En testant le modèle utilisant l'embedding à 70 features, on obtient les mêmes résultats.



Modélisation

Collaborative-Filtering :

- On utilise le **modèle SVD** (technique de factorisation de matrice) de la librairie surprise, qui prend un data contenant 3 colonnes :
 - - l'id du user
 - - l'id de l'article
 - - un "rating" qui correspond au ratio suivant :

$$\text{Rating}(\text{user}, \text{article}) = \frac{\text{Clicks}(\text{user}, \text{article})}{\text{Clicks}(\text{user})}$$

GridSearchCV :

- On va rechercher les meilleurs paramètres du modèle par validation croisée.
- On entraîne le modèle avec les meilleurs paramètres et la totalité du jeu de données.

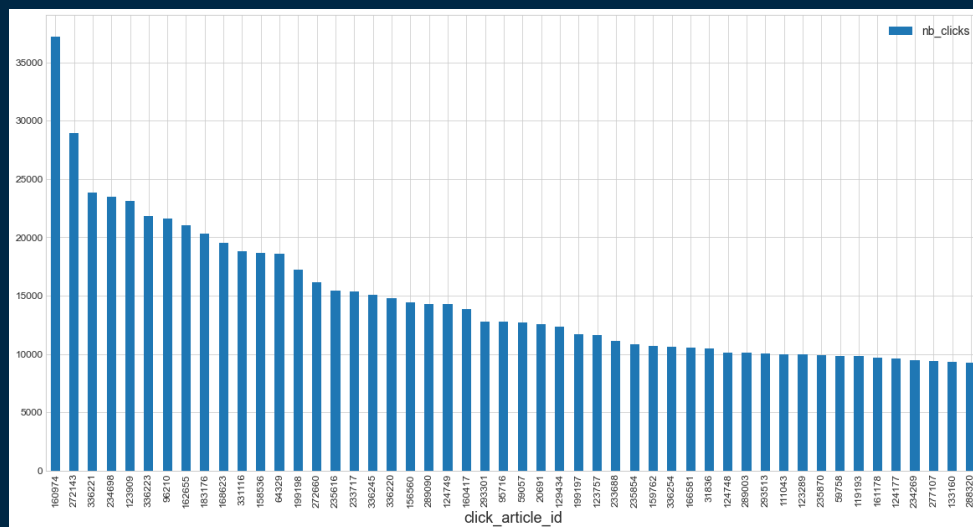
	user_id	article_id	rating
0	0	68866	0.125
1	0	87205	0.125
2	0	87224	0.125
3	0	96755	0.125
4	0	157541	0.125

Modélisation

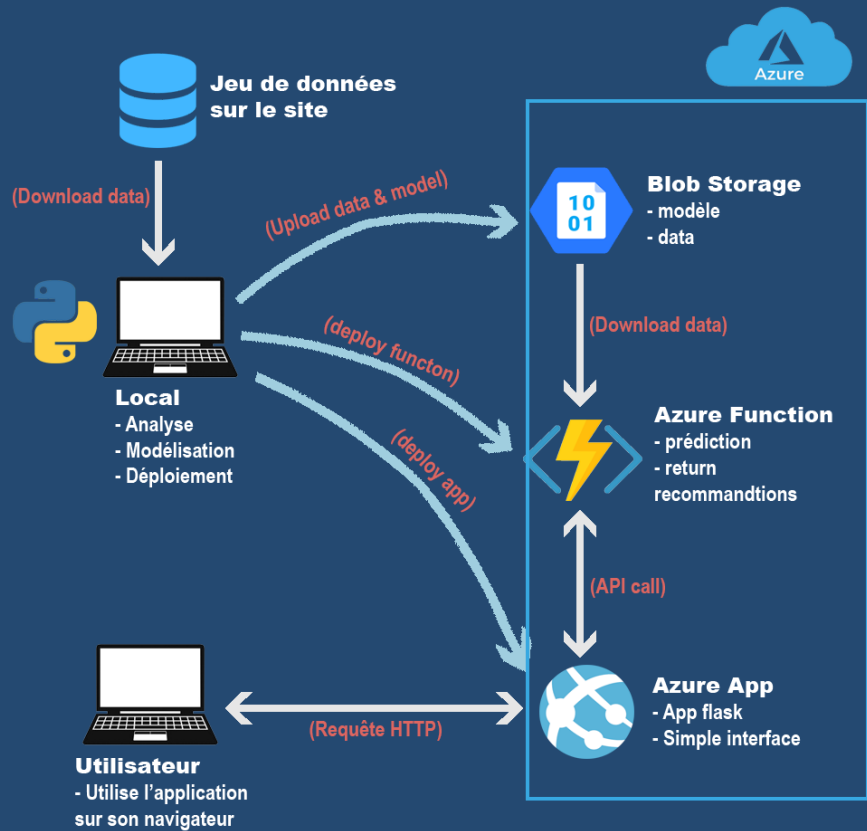
Collaborative-Filtering :

- La prédiction du modèle retourne un score à partir d'un id de user et un id d'article. Ce score est compris entre 0 et 1, et est l'équivalent d'une prédiction du rating calculé précédemment.
- Pour un user on va donc calculer ce score pour chaque article et retourner les n plus élevés.

- On voit que certains articles sont régulièrement recommandés. Il s'agit en fait des articles les plus lus (plusieurs milliers de fois pour certains)



Architecture serverless



Fonctionnalités Azure utilisées :

- **Blob Storage**
- **Azure Function**
- **Azure App**

Déploiement

Stockage sur Azure Blob :

- On va stocker le modèle entraîné ainsi que les données nécessaires à son fonctionnement. Ces données seront ensuite chargées par notre Azure Function.
- L'ACP a permis de réduire la taille de notre jeu de données d'embedding, le stockage étant limité pour l'offre gratuite.



Déploiement sur Azure Function:

- Azure Function est la partie serverless. On va déployer notre fonction, qui effectue la recommandation pour un id de user en utilisant les ressources stockées sur Azure Blob Storage.
- Cette fonction peut ensuite être utilisée via un appel API.



Déploiement

Démo d'Azure Function :

Fonction accessible à l'adresse suivante : <https://ocp9-function.azurewebsites.net/api/recommend-article>

The screenshot displays a REST client interface with the following details:

- Method:** GET
- URL:** `https://ocp9-function.azurewebsites.net/api/recommend-article`
- Buttons:** Send, Cookies, Beautify
- Tabs:** Params, Auth, Headers (8), Body (selected), Pre-req., Tests, Settings
- Body Format:** raw (selected), JSON
- Request Body (JSON):**

```
1 {  
2   "id": 15,  
3   "type": "cb"  
4 }
```
- Response Status:** 200 OK, 800 ms, 250 B
- Response Body (Pretty):**

```
1 [106980, 107298, 107581, 107583, 107299]
```

Déploiement

Déploiement sur Azure App :

- On va ici déployer l'application flask sur Azure App, qui est la partie UI, fournissant une interface utilisateur pour la recommandation d'articles.
- L'application ne possède aucune logique métier, elle se charge simplement d'effectuer l'appel à Azure Function via une requête API, en utilisant les données saisis par l'utilisateur, puis en retournant et en affichant le résultat.



Déploiement



Démo d'Azure App :

Application web accessible à l'adresse suivante : <https://ocp9-recommandation.azurewebsites.net/>

Recommandation d'articles

Choisir un id de user
(entre 0 et 322896)

Choisir le type de recommandation

Soumettre

Articles recommandés pour le user n°15

106980
107298
107581
107583
107299

Recommandation d'articles

Choisir un id de user
(entre 0 et 322896)

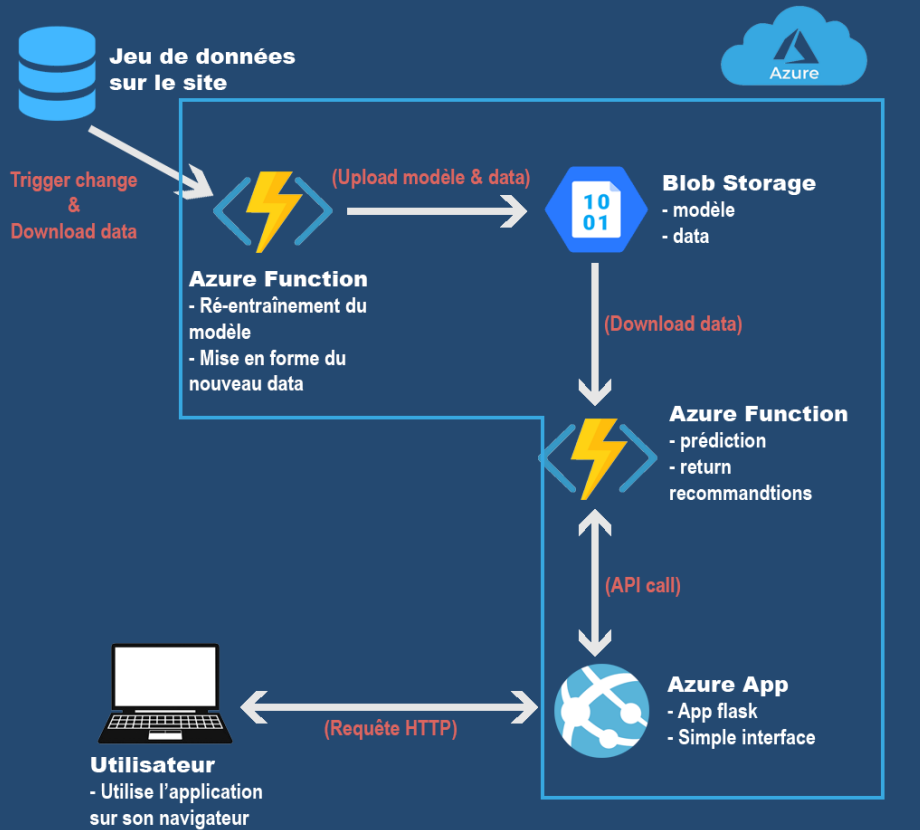
Choisir le type de recommandation

Soumettre

Articles recommandés pour le user n°15

74455
67185
74501
74450
289003

Architecture cible



Architecture cible permettant de prendre en compte la création de nouveaux utilisateurs et de nouveaux articles.

CV - Conclusion



L'application a été conçue comme un **premier MVP**, elle fonctionne et est une bonne base pour venir y greffer des améliorations :

- Créer un **modèle hybride** qui utilise à la fois le Content-Based et le Collaborative Filtering
- Mettre en place **l'architecture cible**, avec une fonction qui détecte un changement dans les données, ré-entraîne un modèle et stock le tout sur Azure Blob.
- Améliorer l'interface de l'application, et sécuriser les échanges.

